APPLICATION FOR REISSUE OF U.S. PATENT NO 5,103,365

1

INTELLIGENT HELP SYSTEM

COPYRIGHT NOTICE

FIELD OF THE INVENTION

This invention relates to help systems for computers: more specifically, it relates to help systems that aid a 15 user of a computer by providing context sensitive help.

BACKGROUND OF THE INVENTION

In order to operate a computer effectively, a user must master a number of commands and data formats. 20 One usually accomplishes this by spending hours reading printed user documentation and/or by using trial and error techniques.

Computer-aided help system have been developed to provide on-line assistance to computer users. In re- 25 sponse to a request by a user, those systems display help information on the display screen of the computer. Simple help systems always start with the same display, regardless of the circumstances, and the user must enter specific information to find help for his or her particular 30 situation. More advanced help systems display context-sensitive help. Context-sensitive help systems determine what particular part of an application program the user is in. Then help information is displayed that is relevant to this user location. 35

While such context-sensitive help systems represent an advancement over simple help systems, they have numerous limitations. Such systems are usually tightly coupled to an application program; they must rely on the application program to keep track of and store the 40 context. Further, since these systems are limited to displaying help information based upon program location, they will always return the same help information for a given location regardless of how the user got there. While such systems provide the convenience of 45 on-line help, the help information they provide is nothing more than a user's manual correlated with a given program screen or function. As a result, these help systems tend to be of limited utility to the user who cannot specifically identify the problem or who has 50 "lost his way."

SUMMARY OF THE INVENTION

The invention recognizes a need for an intelligent help system which processes information specific to the 55 user's history, such as tasks he or she has successfully executed (and how many times) or has had previous help with, and information which defines a state of a machine and a state of a programmed application.

According to the invention, an intelligent help system 60 for aiding the user of a computer program is provided by maintaining an historic queue and using artificial intelligence techniques to select help information based on user-directed events and the current state of the system. In particular, user-directed activities are moni- 65 tored and stored in the historical queue inside a knowledge base. System states are also monitored and stored. The knowledge base is then used by an inference engine

to isolate the specific kind of help that a user needs. Thus, the user is given assistance upon request which is appropriate to that user's level of understanding or experience and the current activities that he or she has
5 executed.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram of a computer system in which the invention may be embodied.
10 FIG. 2 is a block diagram of a computer software system used in the preferred embodiment.

FIG. 3A illustrates the processing of user-directed events and system states into help information.

FIG. 3B is a flow chart of the general methods of the
15 help system.

FIG. 4 is a flow chart of a query by the monitoring device.

FIGS. 5A-B are a flow chart of the methods of the
20 event interpreters.

FIG. 6 is a flow chart of the methods for processing events.

FIG. 7 is a flow chart of the methods for identifying a task.

25 FIG. 8 illustrates the rule taxonomy of the invention.

FIG. 9 illustrates frames and slots for the storage of knowledge.

FIGS. 10A-B are a flow chart of the methods for proving rules.

30 FIG. illustrates the operation of the display engine.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

Referring to FIG. 1, the preferred embodiment of the
35 invention is implemented on a computer system 100 having a central processor 102, a system memory 103, a display device 105, a keyboard 106, a mouse 107, a disk memory 108, an I/O controller 101, and interconnecting means 110, such as a system bus. In the preferred
40 embodiment, a Tandy 1000 series computer (Tandy Corporation of Ft. Worth, Tex.) is used as the system 100.

Referring to FIG. 2, a computer software system 200 is shown for programming the computer system of FIG.
45 1. Software system 200 is stored in system memory 103 and on disk memory 108. System 200 programs the central processor 102 to display a graphic user interface (GUI) on display monitor 105. In the preferred embodiment, help system 204 is implemented in the Tandy
50 DeskMate environment 203 which provides a software interface 205 between a user 206, a computer application 202, and an operating system 201. It will be apparent, however, that one of ordinary skill in the art, informed by this specification, could implement the in-
55 vention in other operating environments.

In the preferred embodiment, an artificial intelligence (AI) paradigm is used to deal with knowledge which may be vast and uncertain, leading to multiple solutions for a given situation. An AI model has the ability to
60 learn or infer more knowledge from what it already knows. Thus, if a user requests help and the help system cannot reach a solution, the system can get further information from the user and then remember the situation; the next time that that situation occurs a solution
65 can be given without querying the user again.

Referring to FIG. 3A, help system 300 of the preferred embodiment comprises a monitoring device 320 for collecting data generated in response to user-

directed events and system states 310. a knowledge base
330 for storing data 331 along with a help information
database 335 used to determine the best help to give, an
inference engine 340 for interpreting data 331 and help
information database 335 in knowledge base 330, and a   5
display engine 350 for presenting appropriate help in-
formation 360 on display device 105. Data 331 com-
prises an historical queue 332 and a state data 333, while
help information database 335 comprises a plurality of
rules 334 and text 336.

FIG. 3B is a flow chart illustrating the general meth-   10
ods of help system 300. In step 351, user-directed events
and system states are monitored. User-directed events
are the activities that a user performs in an application
program, for example, saving a file in a paint application   15
or copying a block of text in a word-processing applica-
tion. The system state comprises a machine state and an
application-specific state. In step 352, the information
collected in step 351 is stored as facts or data 331 of
knowledge base 330. Specifically, sequential user-   20
directed events are stored in historical queue 332 and
knowledge about the system is stored as state data 333.
In step 353 if a user requests help (e.g., pressing F1 key),
then in step 355 inference engine 340 tests known data
331 against help system rules 334. However, if no help   25
is requested in step 353, the routine loops back to the
monitoring step 351.

Knowledge base 330 stores heuristics in the form of
rules. Rules 334, such as those used in step 355, are
premise-conclusion statements predefined by an appli-   30
cation developer which guide inference engine 340 in
selecting an output. For example, suppose a user is in a
Text (word processing) listbox and no files are selected.
A rule that would check this is:

                                                         35
  if
       no files
       running.listbox TEXT
  then
       help running text

                                                         40

This rule attempts to "fire" by proving its premise, "no
files." First, it checks the known data 331 in knowledge
base 330. If this is not in data 331, it checks for other
rules with this premise as their conclusion. Next, if in   45
step 356 a match is found, the corresponding rule fires in
step 357. In step 358, in response to the particular rule
that fired, a conclusion is asserted and appropriate help
information 360 is displayed. The format of rules in this
embodiment is described hereinafter with reference to   50
FIG. 8.

Monitoring device 320 and its functions will now be
described in detail. While monitoring device 320 tracks
or monitors user input, it has processing capabilities. It
may assert data in knowledge base 330 after identifying   55
a sequence of one or more events or states. It may re-
move or retract data concerning states which are no
longer true, or reasserts a new value for old data. It
keeps track of the number of times an activity has suc-
cessfully been completed by a user.   60

Monitoring device 320 monitors different types of
information, including machine states, application spe-
cific states, and historical information. The machine
state includes current system level, such as within an
application or accessory, running a component, or at   65
the desktop interface. An accessory is an application
that may "pop up" over another application. Examples
of accessories include a pop-up calculator, calendar, or

alarm. A component is a graphic element that a user interacts with to display and accept information from a user. For example, the components in a dialogue box include radio buttons, push buttons, and edit fields. The
5 menubar is also a component.

Monitoring device 320 also monitors application-specific states (application specifics), including information unique to an application, and component states, including information within the current machine state
10 which is either application specific or general: since components such as radio buttons are generated in the same way, regardless of whether the component is used at the application level or general level, information about component states is also generated in a uniform
15 manner. For example in all applications which have a menubar, choices are obtained by selecting an item off the menu.

Monitoring device 320 also tracks historical information which indicates the completion of a task for which
20 help has been defined. This data is stored in historical queue 332. A successful completion indicates that the user no longer needs help in performing that task. More specific help information, rather than general information, can be given as the user gains more experience
25 with a program.

The structure of historical queue 332 will now be described in detail. For each entry, an entry type is stored. For example, when a user runs a dialogue box,
30 monitoring device 320 adds a dialogue box entry (CMP_DLG_BOX) into historical queue 332 and then stores corresponding editfield, listbox, radiobutton, iconbutton and checkbox information. Each entry into historical queue 332 can be of variable length, depend-
35 ing on the type of entry made. A unique "return code" is assigned to each component, thus facilitating the distinction between components. A far pointer to an entry's structure (e.g., dialogue box, component, and/or menubar structures) is stored to allow direct access to
40 that structure. Following this, dialogue and component information is copied. The format used can be of variable length. A subentry flag is defined to indicate when there is a subentry. For example, a component may still be running while the menubar is processed, therefore,
45 the menubar is part of a single entry. If the flag is set to a value of true, then any subentry data is stored. Another flag is defined to indicate the user keystrokes that occurred during the entry. This enables the system to determine, for example, whether the user has begun
50 entering data into a dialogue box, or if he chose help immediately. The name of the box and menu entries, such as DLGBOX, MSGBOX, LISTBOX, or MENU, is stored to distinguish one box from another. A third flag is defined to indicate the user keystrokes that oc-
55 curred before invoking help. This enables the system to determine what the user was doing in the application.

Variables which manipulate historical queue 332 are defined as follows:

ihm_HelpQ: the 500 byte queue, there is one in each
60   task data area of the Core Service Routine (CSR).
ihm_TOP: the physical start of the HelpQ buffer.
ihm_ENDQ: the physical end of the HelpQ buffer.
ihm_start_ptr: pointer to the first entry in the queue, (
   which is last historically).
65 ihm_cur_ptr: pointer to the first byte of the last entry
   in the HelpQ.
ihm_end ptr: pointer to the last byte of data entered
   into the HelpQ.

ihm—save—ptr: the ptr value of the last entry made. saved because an invalid ptr is entered into the HelpQ as a NULL.

ihm—menu—ptrl: double word pointer to the current menubar. Updated each time mb draw is called.   5

The data structure of the queue may be summarized as follows:

dw— offset of previous entry

dw— type

dw— return code   10

dw— struct ptr— offset

dw— struct ptr— segment

db— # subentries

db— length of name

db(?)— name of dlg box. msg box. menu. listbox   15

db— keyflag— any mouse or key events before entry?

db(?)— subentry data

db— keyflag— any mouse or key events during entry?

dw— length of information copied or 0   20

In the preferred embodiment. the monitoring of information occurs at different places in software system 200. All state changes that result in the execution of dialogue and message boxes are monitored. In Desk-Mate environment 203. commands are entered by the   25 user through a menubar and are processed by event interpreters. which check for menubar changes. In addition. the application programs themselves may indicate state changes in their respective working areas.

DeskMate environment 203 is divided into the fol-   30 lowing hierarchy of level changes. thus simplifying the monitor's task of updating information:

1. Top Level to application/accessory or menubar/-component:

2. Application to accessory or menubar/component   35 or return to top level;

3. Accessory to menubar/component or return to application or top level.

Specific information is required as to the following states: application is running or has quit; which applica-   40 tion is running; accessory is running or has quit; which accessory is running; dialogue box is running or has quit; message box is running or has quit; component is running or has quit; menubar menu has been pulled down or has returned; user at desk top; if user not at top   45 level. the level attained prior to the current one. The variable "level" takes the value of DeskTop. Menubar. Dlg—box. Message—box. Info—box. or Component. The monitoring can obtain the state changes by getting the address calls from DeskMate environment 203   50 which indicate a function call to an application program or resource.

FIG. 4 illustrates how monitoring device 320 queries for the address of calls it is interested in. In DeskMate environment 203. applications call core service routines   55 (CSR) to run components and dialogue. information. and message boxes. A menubar interpreter handles the processing of the menubar. Therefore. independently of an application. the calls which change the structures that the application is using can be monitored to keep   60 track of information changes.

Thus. the steps are as follows. In step 401. an entry call is made to a core service routine. In step 402. the component type is identified and then compared with a list of components that monitoring device 320 is inter-   65 ested in. In step 403. if the component is listed in the table. then in step 404. monitoring device 320 takes the application's parameters and its structure pointer to

copy data from within the application s structures. The structures are not modified. Upon exit of the service and before control is returned to the application. at step 405 monitoring device 320 again accesses data in the struc-
5 tures. this time for the purpose of historical information updates.

Since a distinction may be made between applications and accessories. a variable "program" is defined to indicate which particular program is running. Program
10 takes the value of the program name. which is the same identifier for help information database 335 associated with the application or accessory.

Two event interpreters pick up menubar changes. The menu selected is important for the state, while the
15 item returned is important for the history. A high-priority event interpreter picks up the menubar changes when help is requested. and a low priority event interpreter will make all changes from the menubar's return code. The high-priority interpreter examines events
20 first before any further processing by the DeskMate system. On the other hand. the low-priority interpreter examines events after they are processed by the Desk-Mate system.

FIGS. 5A-B illustrate the method of the events inter-
25 preters. In step 501. the application registers the menubar with monitoring device 320. At step 502. if the user has selected the F1 (help) key. then at step 503 the state information is updated according to what level the system is operating at (indicated by the variable
30 "level"). If level equals zero in step 504. then the menubar is the last thing changed, therefore a menubar update is needed. At step 505. the menubar interpreter makes an update. But if "level" is not zero. then step 505 is skipped. At step 506. the low priority interpreter
35 checks the event type. If. at step 507. it is menubar (level = menubar). then in step 508 the low priority interpreter does a history update. performed by taking the return code and comparing its value with the menubar menus. The string corresponding to the return code is
40 the item of interest. At step 509. the routine loops back to step 501 to await another F1 keystroke.

The rules associated with the functions of the above components are written such that if a component has a title string, this string is used to identify its help source.
45 If a component does not have a title string. then the name of the component will be used as its identifier.

Since all applications use the features of DeskMate environment 203. standard representation of data 331 and rules 334 is possible. For example. the following
50 variables can be used to indicate events and states:

menu.selected
menu.item.selected
dlg.box.running
dlg.box.focus
55 msg.box.running
cmp.running
listbox.item.selected
checkbox.item.selected

Application-specific information is obtained by hav-
60 ing the application assert data into the current state data 333 in knowledge base 330. The application accomplishes this by making a call to an "Assert" function with the parameters "variable" and "value." which are pointers to strings. The variable should match a variable
65 in the rule premise. i.e.. the rule must be defined beforehand. An application should assert any historical information related to its unique state configuration. Application data are removed from the current state data 333

by the application once those data are no longer true by calling a "Retract" function. Although the applications may directly assert data into knowledge base 330. they do not determine what help to give. Instead. they supply the inputs for this determination. 5

Historical information is obtained by using a user's I.D. to index the user's unique historical information. This historical information is updated at the end of a task completion. For example, when the user has successfully executed a copying function. monitoring device 320 recognizes this by checking that the user has selected "text" and then selected "copy" from the file menu. Having determined that the user has mastered this task. monitoring device 320 updates the user's historical information. For historical information associated with the application's specific data. the application updates the information itself by calling the function UpdateHistory with a parameter pointing to a string representing the activity just completed. 15

FIG. 6 summarizes the method for processing the 20 events from the event interpreters. At step 601. events are processed by examining the mouse coordinates and event type and value. In step 602. events are identified by trying to "fire" or trigger a rule that would make a data assertion or retraction into knowledge base 330. 25 These rules represent all conditions that must be true for data to be asserted or retracted. If a rule fires in step 603. then the event is identified. step 604. If the rule does not fire. then the event is not identified. step 605.

FIG. 7 illustrates the approach used in step 602 (FIG. 30 6) to identify a task (sequence of events). In step 701. a key or mouse event is examined. In step 702. if the event does not match the first premise line in any rule associated with the system level. then the event is discarded in step 704. Otherwise. in step 703. all rules that fire are 35 placed on an Agenda. which represents the most likely task(s). At step 705. the rules are approved or disproved by examining the next key or mouse events that come in. If all the rules in the Agenda fail at step 706. then the first key analyzed is discarded and the next key is used 40 to search for new rules. step 708; otherwise in step 707. the Agenda is cleared and the procedure loops to step 701 to examine the next key/mouse event.

Since the information monitored is dependent upon the machine state level. the data generated are divided 45 according those levels. Monitoring device 320 attempts to assert data which apply to a given level. Each application or accessory can be considered as a distinct object (as a level) which performs certain activities, some of which are common to other objects. Therefore, it is 50 convenient to divide the monitor's data structure into "frames." A frame contains the activities that each application is capable of. with current values and historical information. The frames have slots for storing the data used by the rules or a pointer to another frame. 55

In the preferred embodiment. user input and system state are analyzed by monitoring device 320 for passage as data to knowledge base 330 for storage. Commands are defined for controlling this flow of data to and from knowledge base 330. Data are stored in frames through 60 the Assert command. Old information is deleted with the Retract command. A query is made to knowledge base 330 to attain information through a Query command. Previous data are asserted by a Reassert command. 65

Knowledge base 330 comprises formal (traditional data base information) and informal (heuristic) knowledge which is rule based. FIG. 8 demonstrates the rule

8

taxonomy. At the lower level of the rule hierarchy 800 is a single rule 804 defining a pattern-to-action goal. The pattern is known as the left-hand side of a rule, while the action is the right-hand side. The rules use the common

5 logical operators AND, OR, and NOT, as well as Boolean operators such as IF, THEN, and ELSE. In addition, the key word TEST indicates that a comparison needs to be made. The right-hand side of a rule can have an ELSE clause, an assertion, a retraction, another rule,

10 or a procedure call. For example, in the rule:

    IF a THEN b, ELSE c

b could be an assertion which would add data b into
15 data 331; multiple data assertions are possible. A retraction would delete data b from data 331:

    IF a THEN (RETRACT b).

20 "Another rule" may be imbedded as follows:

    IF a then b, IF (TEST (= bc)) THEN d.

An example of a procedure call would be:

25
    IF a THEN (CALL HelpTutorial(a)).

Linked Rules 803 are a linking between rules which share the same conclusion. This is a design implementa-
30 tion intended to make the inference process easier by knowing alternative solution paths. Rule Groups 802 group rules with a common purpose. For example, rules with a conclusion indicating what specific help to give are all rules determining the goal state. A group can
35 have an (optional) priority identifier so the most important or most specific rules can be tried out first when searching. This does not imply that a rule group will be left out of a search. Rules Classes 801 are a further conceptualization of rules. They separate individual
40 knowledge bases, each having a unique identifier by which it is distinguished. This identifier can be used by a set of control rules which use the machine state information to indicate knowledge base access.

The premise of each rule also contains formal or
45 informal knowledge. The informal knowledge becomes formal when a rule fires successfully. The formal knowledge is stored in and accessed from knowledge base 330 in frame structures. Frames can be made up of other frames, which can also be shared. With this data
50 structure, it is possible to access only the frame associated with the current state information when monitoring device 320 is updating the user's historical information. The frame's slots are like any linked list, except they represent actions and attributes of an object or
55 concepts that the frames represent. Since frames indicate the relationship between a user's activity and the associated heuristics used to interpret that activity, it is easy to access only relevant information. FIG. 9 illustrates a frame 901 with its related slots 902.
60 Rule Classes 801 are made up of pointers to the variables and values in the frames. The slots which contain another frame are really another group within a rule class. In a frame-based reasoning system, one selects the frame to prove by filling in slot values. The slots contain
65 the rules. The successful completion of a frame yields a solution.

BEGIN.RULECLASS

-continued

```
         <id>
         BEGIN.RULEGROUP
         <id>
         IF
         ( <variable> IS <value> )                    5
         ( <value> )
         ( NOT( <variable> IS <value> ))
         ( TEST (<variable> EQ <#> ))
         THEN
         ( <value> )                                  10
         END.RULEGROUP
         END.RULECLASS
```

By way of illustration and not limitation. the rules
may be coded as data structures (illustrated in the C 15
language):

```
/*_____     */
/* Premise is a structure containing a single premise    */
/* of a rule. It is made up of the string itself. the type   */   20
/* of data it is. and the possible values it can hold.    */
/*_____     */
         struct Premise
             {
             char *pVar;
             char *pValueSet;                             25
             char *pValue;
             };
/*_____     */
/* Premises is a structure for maintaining a linked    */
/* list of all the premises (strings) in a single rule    */
/*_____     */   30
         struct Premises
             {
             struct Premise *pPremise;
             int Size;
             struct Premises *pNext;
             };                                           35
/*_____     */
/* RuleClass is a structure of all RuleGroups that can    */
/* associated with one another in searching    */
/*_____     */
         struct RuleClass
             {
             struct RuleGroup *pRuleGroup;                40
             int NumberOfGroups;
             int ID;
             struct RuleClass *pNext
             };
/*_____     */
/* RuleGroup is a structure containing all rules that    */   45
/* are considered to be associated in searching    */
/*_____     */
         struct RuleGroup
             {
             struct Rules *pRules;
             int NumSubGroups;                            50
             int Priority;
             struct RuleGroup *pNext;
             };
.N
/*_____     */
/* Rules is a structure containing all the rules within    */   55
/* a Rule Group. They can be singular or associated    */
/* with one another.    */
/*_____     */
         struct Rules
             {
             struct LinkedRules *pLinkedRules;
             char *pConclusion;                           60
             struct Rules *pNext;
             };
/*_____     */
/* LinkedRules is a structure for maintaining a linked    */
/* list of rules which all have the same conclusion. A    */
/* rule is made up of premises (structure) a conclusion    */   65
/* (string) and a why statement (string) explaining    */
/* why a rule succeeded.    */
/*_____     */
```

-continued

```
struct LinkedRules
    {
        struct Premises *pAllPremises;
        struct LinkedRules *pNext;
    };
/*------------------------------------------------------*/
/* Query is a structure for maintaining a linked list    */
/* of all the queries for information (strings) in the   */
/* knowledge base.                                       */
/*------------------------------------------------------*/
struct Query
    {
        char *pFact;
        char *pAsk;
        char *pSet;
        struct Query *pNext;
    };
```

The inputs to knowledge base 330 are the data collected b monitoring device 320. The outputs of knowledge base 330 are data 331 and rules 334 which inference engine 340 selects for examination and help information text 336 which display engine 350 processes. Rules 334 are predefined by application developer. The frames indicate the relationship between the user's activity and the associated heuristics used to interpret that activity, thus making it easier to access only the information needed. The knowledge-based rules are structured so that obvious associations between rules can be incorporated within knowledge base 330 simplifying the design and the inferences from those rules.

Inference engine 340 interprets data 331 and rules 334 in knowledge base 330 to give a help solution to the user, or it intelligently asks for more information in order to obtain a solution. In the preferred embodiment, inference engine 340 operates using a backward-chaining method. This method starts with a goal state (a particular kind of help) and tries to prove it by reaching initial known data.

As FIGS. 10A-B illustrate, the steps used by inference engine 340 are as follows. In step 1001, knowledge base 330 is accessed according to the system state's level. In step 1002, depending on the system state, a group of rules (goal) is selected from knowledge base 330 and used as an hypothesis. Next in step 1003 an attempt is made to prove each rule's conclusion by proving its premise. If, in step 1004, the premise examined is the conclusion of another rule, it is pushed onto a stack (last-in first-out structure in system memory 103) and an attempt is made to prove the other rule's premise in step 1005. In step 1006, if a premise is proved, it is asserted into a working fact or data list at step 1007. Otherwise if the premise fails, a search is made in step 1008 for a rule with the same conclusion, which inference engine 340 will then try to prove. In step 1009, when all the premises in the rule have been proved, the rule has fired and the conclusion can be asserted at step 1010. In step 1011, this process is repeated until there are no more goal states to analyze or until no solution is found. In the latter case, knowledge base 330 is incomplete, therefore, inference engine 340 queries the user for more information or else explains to the user that no help is defined for the particular scenario.

The inputs to inference engine 340 are data 331 and rules 334 contained in knowledge base 330. Some of rules 334 control other knowledge bases which can be selected. Inference engine 340 first examines rules 334 to select the proper rule class 801. Since the groups 802 under the class 801 indicate if they are goal states, en-

gine 340 can also indicate which rules will be used as hypotheses. Final output from inference engine 340 is a help "tag" that indicates a particular help solution. In addition, the engine creates a temporary output of data asserted while proving the rules.

Display engine 350 is an interface between the output of inference engine 340 and help text 336 giving the user easy access to the most useful help topics. Display engine 350 provides general to in-depth help to a user. It processes the inference engine's help tag to provide context-sensitive help. In addition, the help information itself has tags which are used by display engine 350 to locate further help information. This allows a user to select specific help from a subset of help information.

The help information data structure used by display engine 350 is defined for each application/accessory and system resource interfaces. Help that is repeated across multiple applications is divided into groups. The help information which is given is individualized for novice, intermediate, and advanced users. A display format is used which allows a user to select a single item of help from a suggested list, allow the user to continue selecting help until he wants to quit, and allow the user to search through the Help Topics for a particular topic.

The structure by which the help information is accessed and stored is classified according to application-/accessory or system component (requires specific name identification, not type), experience level of user, kind (autodetect or user-invoked), and topic identifier. For example, a help structure (in C language) can be:

```
struct Help Source {
    int     Subject;
    int     Kind;
    int     HelpLevel;
    char    *pTopicString };
```

which can be filled by:

```
            Subject  equal to TEXT_SUBSTITUTION
    Kind        equal to AUTO
```

and pTopicString pointing to "Text Substitution."

Referring to FIG. 11, the processing of help information in display engine 350 will now be described in detail. Help information database 335 is a database containing text 336 and rules 334 fields, i.e., help information text 336 is linked to help information rules 334. In addition, a tag 1102 is provided for representing the solution that a rule produces. Display engine 350 matches tag 1102 with a solution tag 1101 from inference engine 340. The corresponding text (from text 336) is the actual text sent as help information 360 to display device 105.

The organization of rules 334 is as follows. It contains the following fields: Grp, Rule#, PorC#, Var#, Var, Val, Bind, KeyW, and Q#. Grp is a rule group. Group 0 is always the group in which rules are placed that will give a solution. If these rules cause any other rules to fire, these rules are in another group. Rule# is the number of the rule. The rules are sorted by rule number, because one may want to try to fire one rule before another. PorC# is a premise or conclusion number. The records in the table are also sorted by PorC#. The conclusion is number 0 because it is the first thing pulled out of the table when inference engine 340 starts to fire